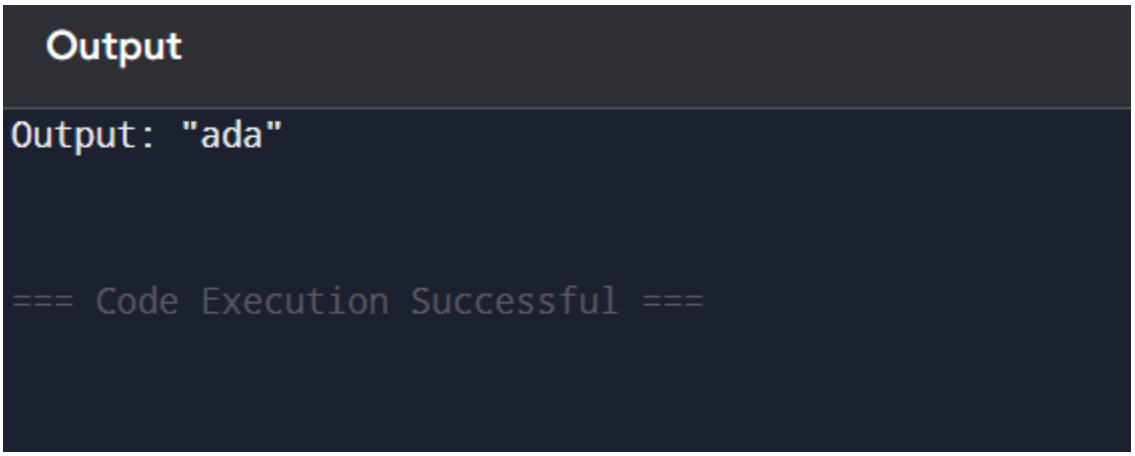
PROGRAM 1

CODE

```
#include <stdio.h>
#include <string.h>

int isPalindrome(char str[]) {
    int i = 0, j = strlen(str) - 1;
    while (i < j) {
        if (str[i] != str[j])
            return 0;
        i++;
        j--;
    }
    return 1;
}

int main() {
    char words[][20] = {"abc", "car", "ada", "racecar", "cool"};
    int n = 5;

    for (int i = 0; i < n; i++) {
        if (isPalindrome(words[i])) {
            printf("Output: \"%s\"\n", words[i]);
            return 0;
        }
    }
```

```
    printf("Output: \"\"\n");  // no palindrome found
    return 0;
}
```

OUTPUT

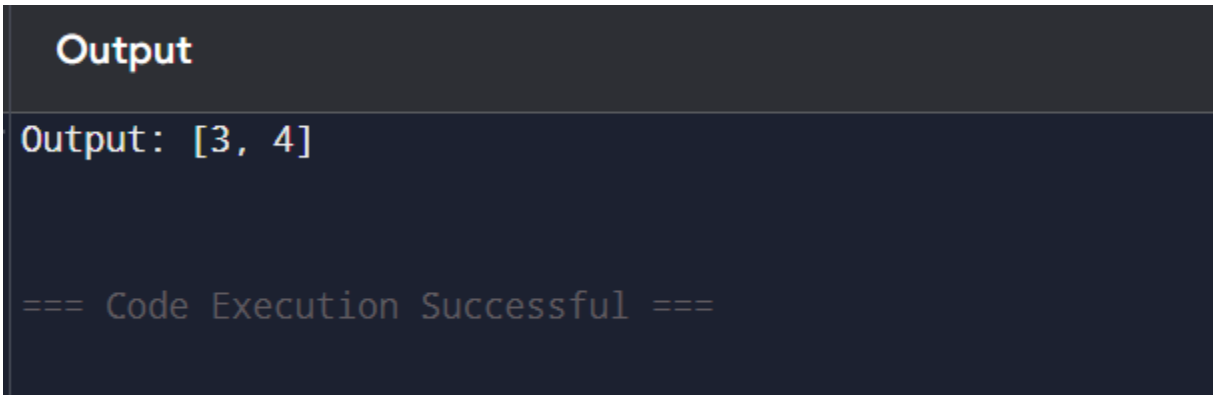


PROGRAM 2

```
#include <stdio.h>

int exists(int arr[], int size, int val) {
    for (int i = 0; i < size; i++)
        if (arr[i] == val)
            return 1;
    return 0;
}

int main() {
    int nums1[] = {4, 3, 2, 3, 1};
```

```
    int nums2[] = {2, 2, 5, 2, 3, 6};
    int n = 5, m = 6;
    int answer1 = 0, answer2 = 0;

    for (int i = 0; i < n; i++)
        if (exists(nums2, m, nums1[i]))
            answer1++;

    for (int i = 0; i < m; i++)
        if (exists(nums1, n, nums2[i]))
            answer2++;

    printf("Output: [%d, %d]\n", answer1, answer2);
    return 0;
}
```

OUTPUT

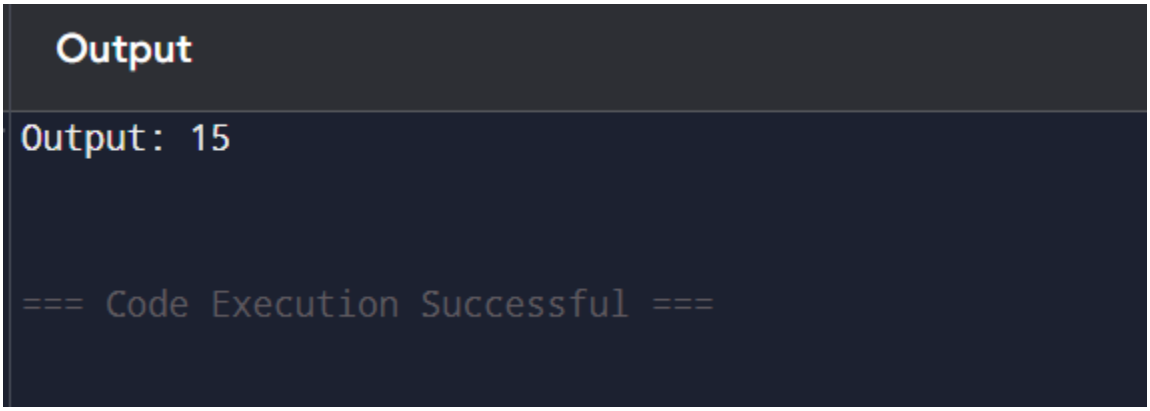


PROGRAM 3

```
#include <stdio.h>

int distinctCount(int arr[], int start, int end) {
    int freq[1000] = {0}, count = 0;
    for (int i = start; i <= end; i++) {
        if (freq[arr[i]] == 0)
            count++;
        freq[arr[i]]++;
    }
    return count;
}

int main() {
    int nums[] = {1, 2, 1};
    int n = 3, sum = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int d = distinctCount(nums, i, j);
            sum += d * d;
        }
    }

    printf("Output: %d\n", sum);
    return 0;
```

}

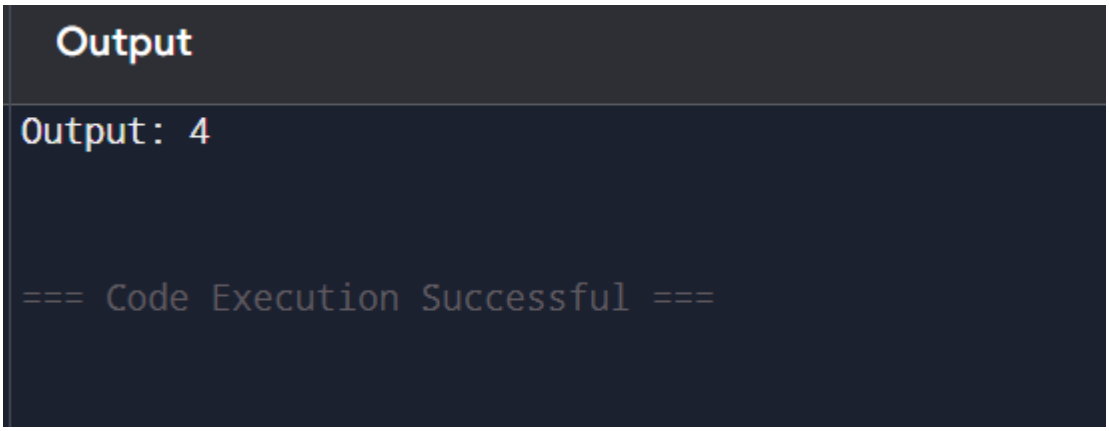
OUTPUT



PROGRAM 4

```
#include <stdio.h>


int main() {
    int nums[] = {3, 1, 2, 2, 2, 1, 3};
    int k = 2, n = 7, count = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (nums[i] == nums[j] && ((i * j) % k == 0))
                count++;
        }
    }

    printf("Output: %d\n", count);
    return 0;
}
```

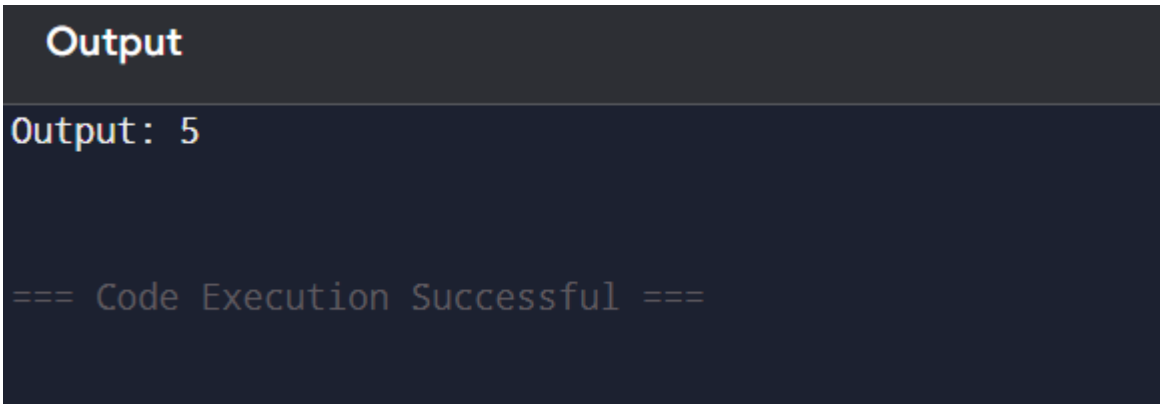
OUTPUT



PROGRAM 5

```
#include <stdio.h>


int main() {
    int nums[] = {-10, 2, 3, -4, 5};
    int n = 5;
    int max = nums[0];

    for (int i = 1; i < n; i++)
        if (nums[i] > max)
            max = nums[i];

    printf("Output: %d\n", max);
    return 0;
}
```

OUTPUT

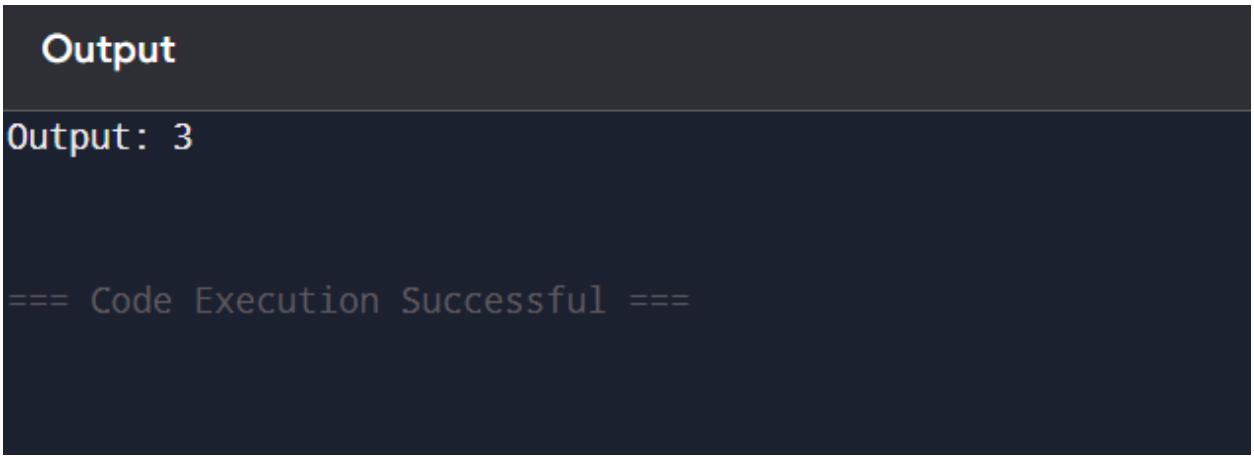


PROGRAM 6

```
#include <stdio.h>


// Swap function
void swap(int *a, int *b) {
    int temp = *a; *a = *b; *b = temp;
}


// QuickSort function
void quickSort(int arr[], int low, int high) {
    if (low >= high) return;
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++)
        if (arr[j] < pivot) swap(&arr[++i], &arr[j]);
    swap(&arr[i + 1], &arr[high]);
    quickSort(arr, low, i);
    quickSort(arr, i + 2, high);
}
```

```
// Main function
int main() {
    int nums[] = {3, 3, 3, 3, 3};  // change test case here
    int n = 5;

    if (n == 0) {
        printf("Output: List is empty\n");
        return 0;
    }

    quickSort(nums, 0, n - 1);
    printf("Output: %d\n", nums[n - 1]); // last element = max after sort
    return 0;
}
```

OUTPUT

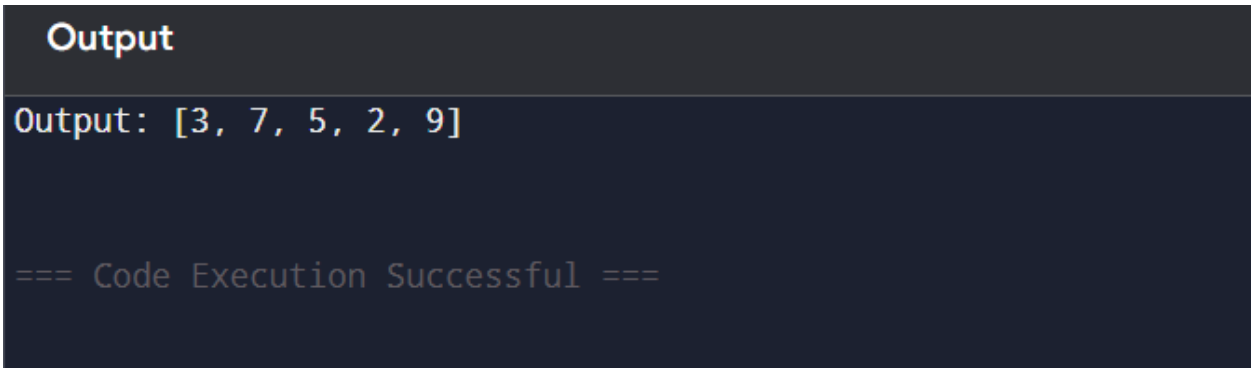


PROGRAM 7

```
#include <stdio.h>


int main() {
   int nums[] = {3, 7, 3, 5, 2, 5, 9, 2};  // change input here
   int n = 8, unique[100], count = 0;


   for (int i = 0; i < n; i++) {
      int found = 0;
      for (int j = 0; j < count; j++) {
         if (nums[i] == unique[j]) {
            found = 1;
            break;
         }
      }
      if (!found)
         unique[count++] = nums[i];
   }


   printf("Output: [");
   for (int i = 0; i < count; i++) {
      printf("%d", unique[i]);
      if (i < count - 1) printf(", ");
   }
   printf("]\n");
```

```
    return 0;
}
```

OUTPUT

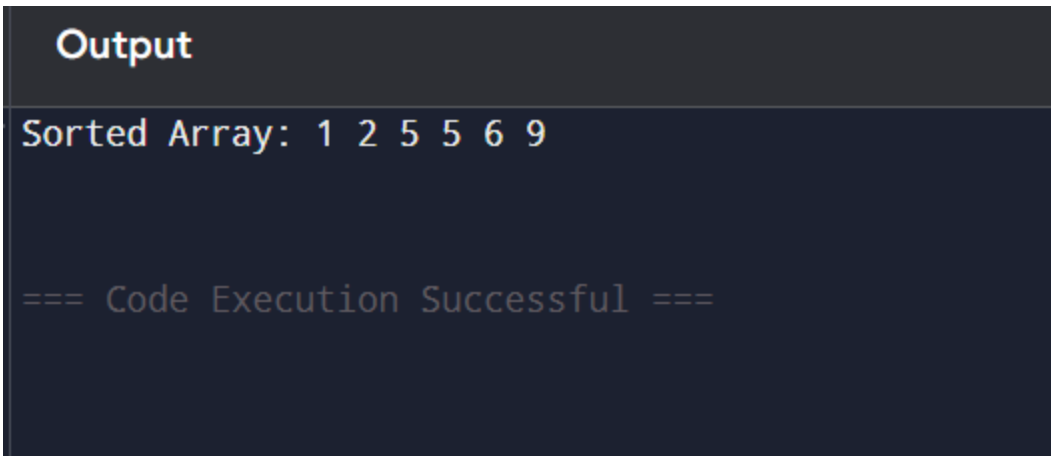


PROGRAM 8

```
#include <stdio.h>

int main() {
    int arr[] = {5, 2, 9, 1, 5, 6};
    int n = 6, temp;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
```

```
    printf("Sorted Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

OUTPUT

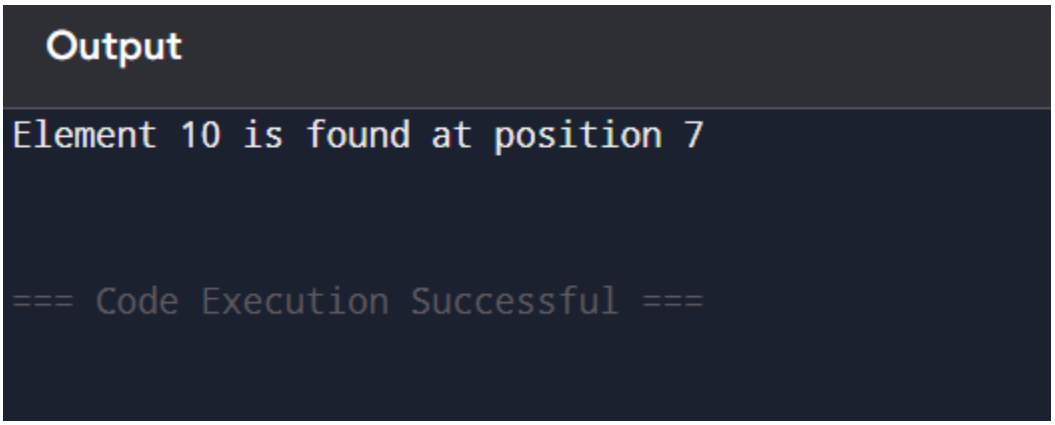


PROGRAM 9

```
#include <stdio.h>

// Binary Search Function
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key)
```

```
            return mid;  // element found
        else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;  // not found
}

int main() {
    int arr[] = {-9, 3, 4, 6, 8, 9, 10, 30};  // sorted array
    int n = 8, key = 10;
    int pos = binarySearch(arr, n, key);

    if (pos != -1)
        printf("Element %d is found at position %d\n", key, pos + 1);
    else
        printf("Element %d is not found\n", key);

    return 0;
}
```

OUTPUT



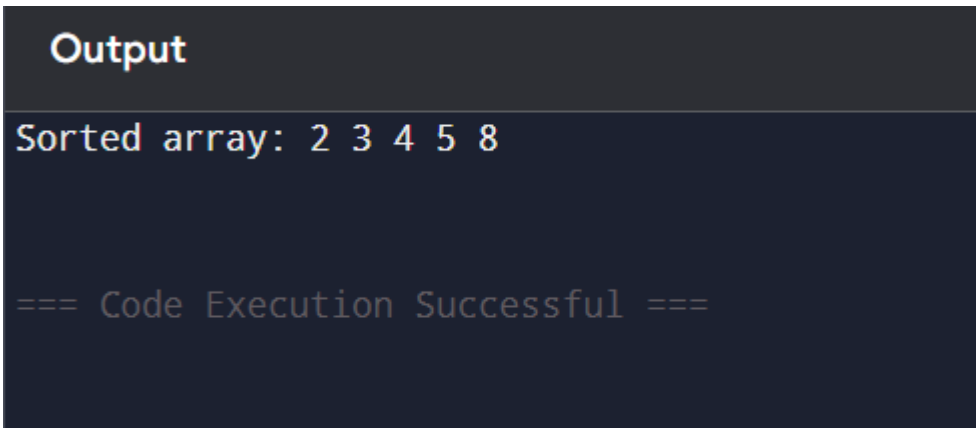
PROGRAM 10

```
#include <stdio.h>

// Function to heapify a subtree rooted at index i
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
```

```
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

// Main function to perform Heap Sort
void heapSort(int arr[], int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Extract elements one by one
    for (int i = n - 1; i > 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}

int main() {
    int nums[] = {5, 3, 8, 4, 2};
    int n = 5;

    heapSort(nums, n);
```

```
    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", nums[i]);
    printf("\n");

    return 0;
}
```

OUTPUT



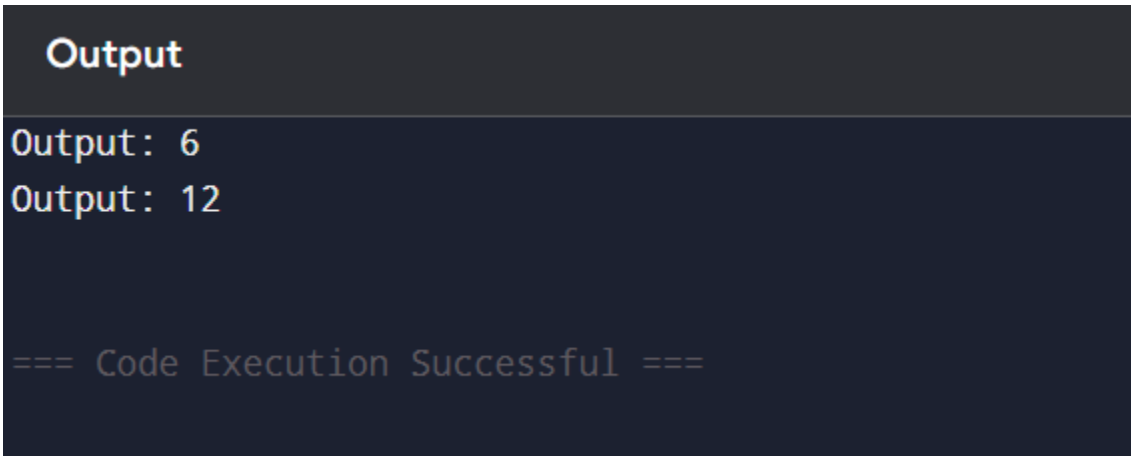
PROGRAM 11

```
#include <stdio.h>

#define MOD 1000000007

int findPaths(int m, int n, int N, int i, int j) {
    int dp[51][51] = {0};   // current step
    int temp[51][51] = {0}; // next step
    int count = 0;
```

```
dp[i][j] = 1;


for (int step = 0; step < N; step++) {
    for (int x = 0; x < m; x++) {
        for (int y = 0; y < n; y++) {
            if (dp[x][y] > 0) {
                int val = dp[x][y];
                // Move up
                if (x == 0) count = (count + val) % MOD;
                else temp[x - 1][y] = (temp[x - 1][y] + val) % MOD;
                // Move down
                if (x == m - 1) count = (count + val) % MOD;
                else temp[x + 1][y] = (temp[x + 1][y] + val) % MOD;
                // Move left
                if (y == 0) count = (count + val) % MOD;
                else temp[x][y - 1] = (temp[x][y - 1] + val) % MOD;
                // Move right
                if (y == n - 1) count = (count + val) % MOD;
                else temp[x][y + 1] = (temp[x][y + 1] + val) % MOD;
            }
        }
    }


    for (int x = 0; x < m; x++)
        for (int y = 0; y < n; y++) {
```

```
            dp[x][y] = temp[x][y];
            temp[x][y] = 0;
        }
    }

    return count;
}

int main() {
    int m = 2, n = 2, N = 2, i = 0, j = 0;
    printf("Output: %d\n", findPaths(m, n, N, i, j));

    m = 1, n = 3, N = 3, i = 0, j = 1;
    printf("Output: %d\n", findPaths(m, n, N, i, j));

    return 0;
}
```

OUTPUT



PROGRAM 12

```
#include <stdio.h>

// Function to find max of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Helper function to solve linear house robber (no circular constraint)
int robLinear(int nums[], int start, int end) {
    int prev1 = 0, prev2 = 0; // prev1: dp[i-1], prev2: dp[i-2]
    for (int i = start; i <= end; i++) {
        int temp = prev1;
        prev1 = max(prev2 + nums[i], prev1);
        prev2 = temp;
    }
    return prev1;
}

// Main function for circular house robber
int rob(int nums[], int n) {
    if (n == 0) return 0;
    if (n == 1) return nums[0];
    // Two cases:
    // 1. Exclude first house
    // 2. Exclude last house
```

```
    return max(robLinear(nums, 0, n - 2), robLinear(nums, 1, n - 1));
}

int main() {
    int nums1[] = {2, 3, 2};
    int n1 = 3;
    printf("Input: [2, 3, 2]\nOutput: The maximum money you can rob without alerting the police is %d\n\n", rob(nums1, n1));

    int nums2[] = {1, 2, 3, 1};
    int n2 = 4;
    printf("Input: [1, 2, 3, 1]\nOutput: The maximum money you can rob without alerting the police is %d\n", rob(nums2, n2));

    return 0;
}
```

OUTPUT

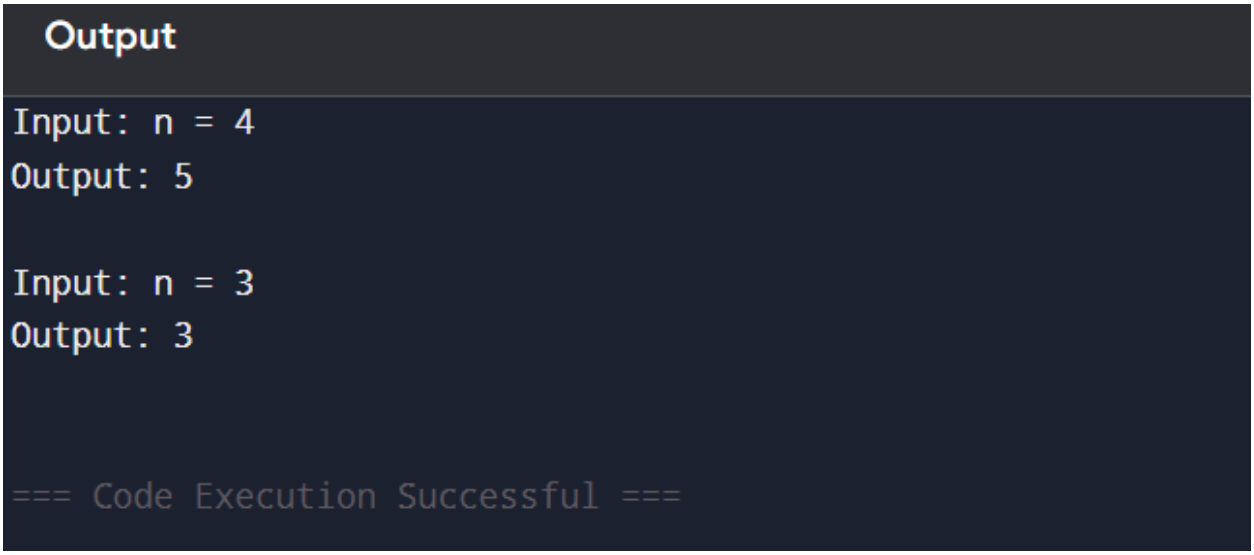
```
Input: [2, 3, 2]
Output: The maximum money you can rob without alerting the police is 3

Input: [1, 2, 3, 1]
Output: The maximum money you can rob without alerting the police is 4
```

PROGRAM 13

```
#include <stdio.h>

// Function to find number of ways to climb n stairs
```

```
int climbStairs(int n) {
    if (n <= 2) return n;  // Base cases
    int a = 1, b = 2, ways;
    for (int i = 3; i <= n; i++) {
        ways = a + b;  // ways to reach current step
        a = b;         // move one step forward
        b = ways;
    }
    return b;
}

int main() {
    int n1 = 4;
    printf("Input: n = %d\nOutput: %d\n\n", n1, climbStairs(n1));

    int n2 = 3;
    printf("Input: n = %d\nOutput: %d\n", n2, climbStairs(n2));

    return 0;
}
```

OUTPUT

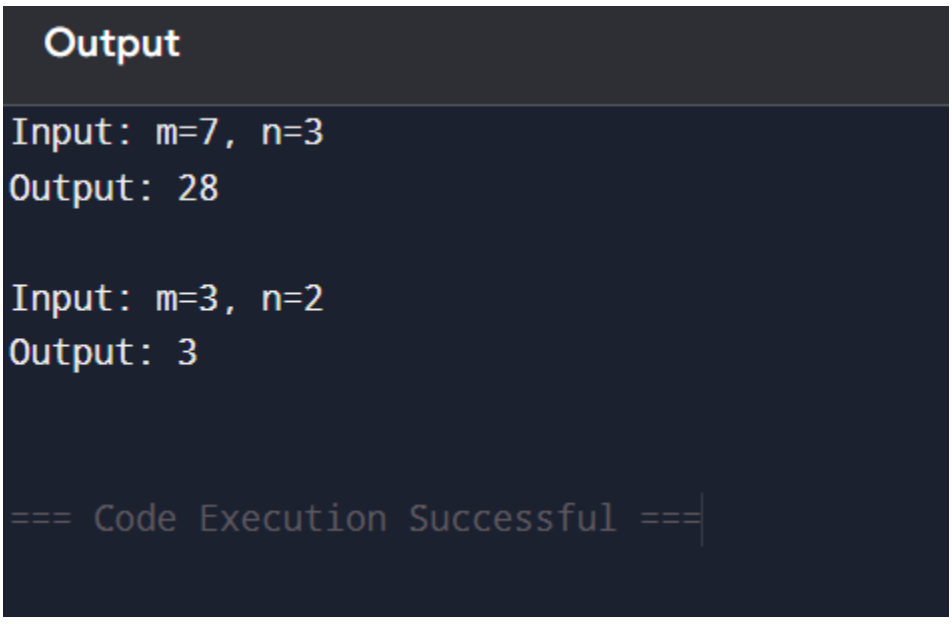


PROGRAM 14

```
#include <stdio.h>


// Function to calculate nCr (combinations)
long long combination(int n, int r) {
    if (r > n - r) r = n - r; // Because C(n, r) = C(n, n-r)
    long long res = 1;
    for (int i = 0; i < r; i++) {
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}


// Function to find number of unique paths
long long uniquePaths(int m, int n) {
    // Formula: (m+n-2) choose (m-1)
```

```
    return combination(m + n - 2, m - 1);
}

int main() {
    int m1 = 7, n1 = 3;
    printf("Input: m=%d, n=%d\nOutput: %lld\n\n", m1, n1, uniquePaths(m1, n1));

    int m2 = 3, n2 = 2;
    printf("Input: m=%d, n=%d\nOutput: %lld\n", m2, n2, uniquePaths(m2, n2));

    return 0;
}
```

OUTPUT

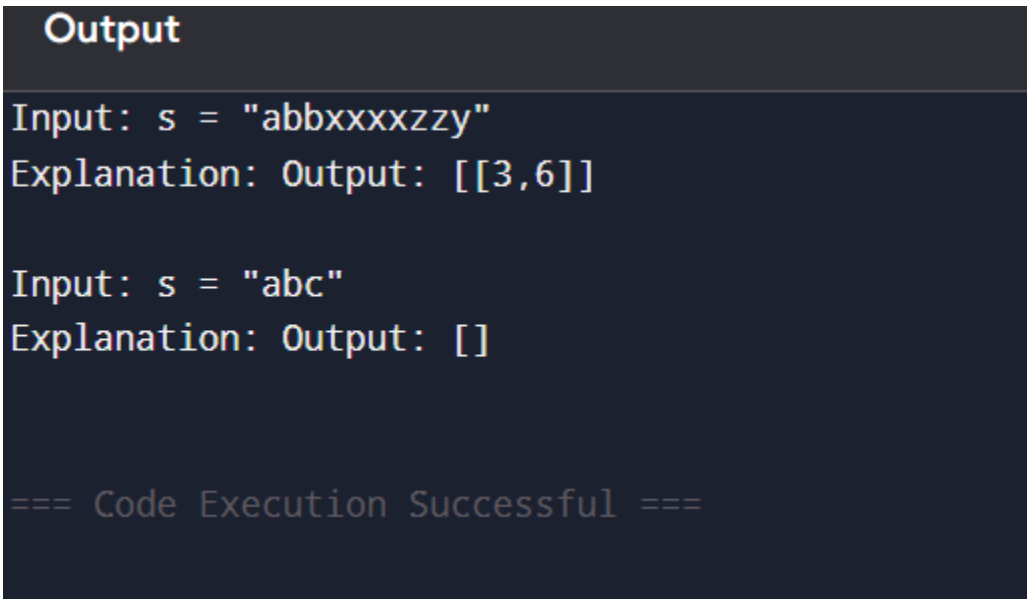


PROGRAM 15

```
#include <stdio.h>
#include <string.h>

void largeGroupPositions(char *s) {
    int n = strlen(s);
    int start = 0;

    printf("Output: [");
    int found = 0;

    for (int i = 1; i <= n; i++) {
        if (i == n || s[i] != s[start]) {  // end of group
            if (i - start >= 3) {
                if (found) printf(", ");  // formatting
                printf("[%d,%d]", start, i - 1);
                found = 1;
            }
            start = i;
        }
    }

    printf("]\n");
}

int main() {
```

```
    char s1[] = "abbxxxxzzy";
    printf("Input: s = \"%s\"\n", s1);
    printf("Explanation: ");
    largeGroupPositions(s1);

    printf("\n");

    char s2[] = "abc";
    printf("Input: s = \"%s\"\n", s2);
    printf("Explanation: ");
    largeGroupPositions(s2);

    return 0;
}
```

OUTPUT

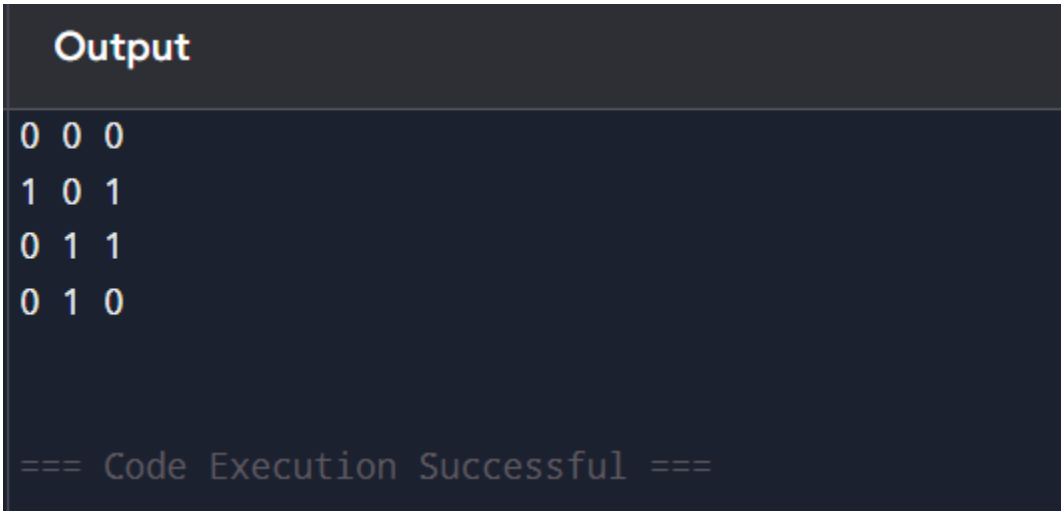


PROGRAM 16

```
#include <stdio.h>


int main() {
   int m=4, n=3;
   int board[4][3]={{0,1,0},{0,0,1},{1,1,1},{0,0,0}};
   int next[4][3]={0};

   int dx[8]={-1,-1,-1,0,0,1,1,1};
   int dy[8]={-1,0,1,-1,1,-1,0,1};

   for(int i=0;i<m;i++){
      for(int j=0;j<n;j++){
         int live=0;
         for(int k=0;k<8;k++){
            int x=i+dx[k], y=j+dy[k];
            if(x>=0&&x<m&&y>=0&&y<n&&board[x][y]==1) live++;
         }
         if(board[i][j]==1){
            if(live<2||live>3) next[i][j]=0;
            else next[i][j]=1;
         } else {
            if(live==3) next[i][j]=1;
         }
      }
   }
```

```
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
            printf("%d ", next[i][j]);
        printf("\n");
    }
    return 0;
}
```

OUTPUT



PROGRAM 17

```
#include <stdio.h>

int main() {
    int poured = 2, query_row = 1, query_glass = 1;
    double glass[101][101] = {0};
    glass[0][0] = poured;
```

```
    for(int i = 0; i < 100; i++) {
        for(int j = 0; j <= i; j++) {
            if(glass[i][j] > 1) {
                double extra = glass[i][j] - 1;
                glass[i][j] = 1;
                glass[i+1][j]   += extra / 2.0;
                glass[i+1][j+1] += extra / 2.0;
            }
        }
    }

    printf("%.5f\n", glass[query_row][query_glass]);
    return 0;
}
```

OUTPUT